iterate	returns the prefetch address of the active request register.
result	moves the active request to the result queue.
replace	replaces the prefetch address in the active request field and moves the active request to the prefetch issued queue.
key	returns the search key value of the active request.
request	returns the request id of the the active request.
extract	returns the request at the head of the result queue.

Table 2

Claims

Having described and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention can be modified in arrangement and detail without departing from such principles. I claim all modifications and variations coming within the spirit and scope of the following claims.

I claim:

1. A method of scheduling units of work for execution on a computer system including a data cache or data cache hierarchy, the method comprising the steps of:
   buffering a plurality of transactions on a data structure;
   scheduling a plurality of said transactions on said data structures in a loop;
   issuing prefetch instructions within the body of said loop for the data required to process said transactions.
2. The method of buffering the results of the transactions processed on a computer system in accordance with claim 1, wherein the results are buffered as well, thereby allowing multiple results to be processed together at a later time.
3. The method of processing the results of a completed traversal on a data structure on a computer system according to claim 1 once a traversal of the data structure has completed.
4. The method of associating a request identifier with each transaction on a data structure represented on a computer system according to claim 1 so as to process requests at a time when the number of buffered transactions has reached a threshold at which software pipelined prefetching across the accumulated set of transactions can be applied in sufficient number so that the cumulative gains outweigh the inherent overhead of the method.
5. The method of associating a prefetch descriptor with each data structure that describes the invariants across buffered requests on a computer system according to claim 1, where the invariants include the pipeline depth (D), the startup threshold (K), and the optional completion threshold (Z), optionally the size of the prefetch target at each request, and optionally a small buffer for application specific data.

6. The method of initiating execution of the software pipeline loop on a computer system according to claim 1 once the number of accumulated requests has reached a startup threshold (K).
7. The method of allowing a computer system according to claim 6 to proceed with processing any buffered transactions before the startup threshold (K) has been reached.
8. The method of exiting the software pipeline on a computer system according to claim 1 when the number of unprocessed transactions buffered according to claim 1 reaches a completion threshold (Z).
9. The method of buffering the transaction results on a computer system according to claim 1 whereby a completed transaction is swapped with a transaction that has not yet completed, thereby eliminating the need for additional buffer space.
10. The method of buffering the transaction results on a computer system according to claim 1 whereby the completed transactions are maintained in a FIFO.
11. The method of selecting the next node to prefetch in the software pipeline executing on a computer system according to claim 1 whereby a transactions is selected from the set of buffered transactions if a transaction on the given data structure has been completed, and the next traversal node in the data structure is prefetched otherwise.
12. The method of forcing the completion of the requests buffered in a computer system according to claim 1, thereby ensuring that the time required to complete any buffered transaction can be bounded, and allowing the computer system to complete buffered traversal requests when it might otherwise be idle.
13. A computer system with a cache hierarchy comprising:
    a) at least one main memory,
    b) at least one cache coupled to main memory,
    c) a means for prefetching data into any such cache from main memory,
    d) a buffer for accumulating traversal requests,
    e) a buffer for storing traversal results,
    f) a means of storing traversal requests once prefetch operations have been initiated,
    g) a buffer for holding an active traversal request,
    h) a multiplexor to select between the accumulated traversals and the active traversal.
14. A cache memory system according to claim 13 wherein a prefetch control word is maintained which describes the prefetch target in terms of software pipeline depth, completion threshold, startup threshold, a real-time timeout value, a sequence of control bits that specify the handling of timer events, a prefetch target descriptor, said prefetch target descriptor providing a description to the prefetch unit of the number and stride of words to be prefetched relative to the prefetch target specified as part of the traversal request when a plurality of cache lines to

be prefetched are associated with each prefetch target address, and a mode field that distinguishes between different interpretations of the prefetch target descriptor fields.

15. A cache memory system according to claim 13 wherein a buffer is used to store a representation of traversal requests for which an associated prefetch request has been issued.
16. A cache memory system according to claim 15 wherein said buffer is implemented as a queue.
17. A cache memory system according to claim 13 wherein the traversal request is represented by an address or request identifier, an application supplied value such as a key, and the address of a node in the data structure to be traversed.
18. A cache memory system according to claim 13 wherein a buffer holds the traversal request for which a data structure traversal is in progress.
19. A cache memory system according to claim 18 including a means of reading the contents of subfields of said buffer, a means of storing and extracting subfields of said buffer.
20. A cache memory system according to claim 18 wherein a next register (N) is provided whereby writing to said device register causes the active traversal request address field to be updated with the value written to said device register, the active traversal request to be added to the prefetch issued queue, and a prefetch issued for the device according to the specifications of the prefetch control register.
21. A cache memory system according to claim 18 wherein a result register (R) is provided where, upon to said register being updated: the active traversal request address field is updated to the value written to said device register; if an active results buffer is employed, the active traversal request is added to the results buffer; a traversal request from the accumulation buffer added to the prefetch issued queue, and a prefetch issued for the prefetch address specified by the prefetch issued buffer.
22. A cache memory system according to claim 18 wherein a current register (C) is associated with the prefetch issued queue, reading said register triggers causes the head of the prefetch issued queue to be dequeued into the active request buffer.
23. A cache memory system according to claim 13 wherein a buffer stores completed data structure traversal requests.
24. A cache memory system according to claim 23, which provides a means of removing a traversal request from said buffer.
25. A cache memory system according to claim 13, wherein access to any of said buffers is provided by means of memory mapped device interfaces.

26. The method of organizing data within the memory on a computer system, the method comprising the steps of:
    a) determining the cache line boundaries of data structure elements;
    b) aligning the base of the data structure on a cache line boundary;
    c) homogenizing the data structure;
    d) inserting a pad field into data structure elements so that subsequent elements are aligned on cache line boundaries;
    e) packing elements so as to maximize the data represented in each cache line by removing pointers to adjacent elements, whereby the program instructions that traverse the data structure are constructed to traverse the adjacent packed elements before traversing non-packed elements,

    whereby steps b, c, d, and e may be performed in any order and any proper subset of steps c, d, and e can be employed.
27. The method of creating a homogeneous hash table according to claim 26 whereby the hash function directly indexes an array of nodes in the hash chain, rather than an array of pointers to hash chain nodes, thereby decreasing the number of memory references required to traverse the hash bucket chain, and therefore potential data cache misses, by one.
28. The method of creating a graph represented as adjacency lists according to claim 26 whereby the nodes in the adjacency list are aligned on cache line boundaries, padded, and packed.